[SYSTEM-DESIGN]

Designing Data-Intensive Applications – Chapter 2: Data Models and Query Languages

Posted by CHARLES *on* 2020-04-02

《Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems》 https://amzn.to/2WYphy6

- **Relational DB:** MS SQL, MySQL, IBM DB2, PostgreSQL, SQLite etc.
- **Document DB:** Cassandra, HBase, Google Spanner, RethinkDB, MongoDB etc.
- **Graph DB**: Neo4j,Titan,InfiniteGraph,AllegroGraph,Cypher,SPARQL,Gremlin,Pregel
- **Data Models**:
  - Not only on how the software is written, but also on how we think about the problem that we are solving.
  - Each layer hides the complexity of the layers below it by providing a clean data model.

Relational Model vs. Document Model:

- **Relational Model**: data is organized into relations (called tables in SQL), where each relation is an unordered collection of tuples (rows in SQL)
- **Birth of NoSQL**: (high write)
  - A need for **greater scalability** than relational databases can easily achieve, including very large datasets or very **high write throughput**.
  - **Polyglot persistence**: Hybrid → SQL + NoSQL
- **The Object-Relational Mismatch**: The disconnect between the models is sometimes called an impedance mismatch.
- **Many-to-One and Many-to-Many Relationships**:
  - When you store the text directly, you are duplicating the human-meaningful information in every record that uses it.
  - Removing such duplication is the key idea behind **normalization** in databases.

    - You only need to build a query optimizer once, and then all applications that use the database can benefit from it.
    - Original **Document DB** is Good for One-To-Many, but Not good for Many-to-One;
- **Relational vs. Document Databases Today(Data Model)**:
    - **Document DB vs. Relational DB**
        - **Document DB**: The main arguments in favor of the document data model are **schema flexibility**, **better performance due to locality**, and that for some applications it is closer to the data structures used by the application.
        - **Relational DB**: The relational model counters by providing b**etter support for joins**, and many-to-one and many-to-many relationships.
    - **Which data model leads to simpler application code**? it depends on the kinds of relationships that exist between data items.
        - If the data in your application has a document-like structure (i.e., a tree of one-to-many relationships, where typically the entire tree is loaded at once), then it's probably a good idea to use a document model.
            - **Limitation**: deeply nested; joins; many-to-many
    - **Schema flexibility in the document model**:
        - **Schema-on-read** (DocumentDB, e.g. dynamic runtime type checking) vs. **Schema-on-write** (Relational DB, e.g. static compile time type checking)
            - E.g. default of NULL and fill it in at read time, like it would with a document database.
    - **Data locality for queries**:
        - If your application often needs to access the entire document (for example, to render it on a web page), there is a performance advantage to this *storage locality*.
        - The idea of grouping related data together for locality is not limited to the document model. (e.g. Spanner DB, Oracle, Bigtable)
    - **Convergence of document and relational databases**:
        - It seems that relational and document databases are becoming more similar over time, and that is a good thing: the data models complement each other.

Query Languages for Data:

- SQL is a *declarative* query language, whereas IMS and CODASYL query the database using *imperative* code.
    - **Declarative** query language: hides implementation details of the database engine; often lend themselves to parallel execution.(which is important)
    - **Declarative Queries on the Web**:
        - E.g. CSS/XSL vs. DOM API
    - **MapReduce Querying**: MapReduce is a programming model for processing large amounts of data in bulk across many machines, popularized by Google. (e.g. MongoDB)

Graph-Like Data Models:

- A graph consists of two kinds of objects: vertices (also known as nodes or entities) and edges (also known as relationships or arcs).

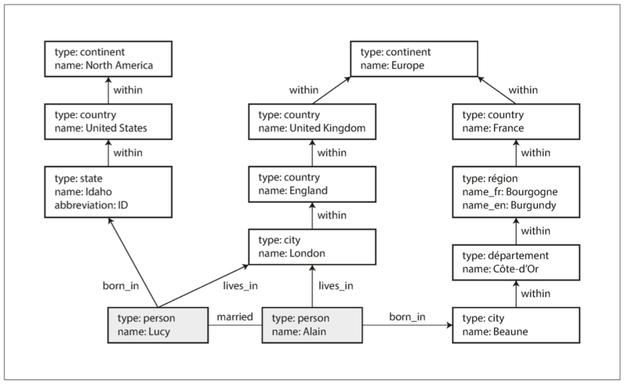

Figure 2-5. Example of graph-structured data (boxes represent vertices, arrows represent edges).

- **Good for**: If your application has mostly one-to-many relationships (tree-structured data) or no relationships between records.
- **Property graph** model (implemented by Neo4j, Titan, and InfiniteGraph): No schema restricts; travers; maintaining a clean model;
  - **Cypher Query Language(**Neo4j**)**: declarative query language for property graphs
  - **Graph Queries in SQL**: possible but difficult;
- **Triple-store** model (implemented by Datomic, AllegroGraph, and others): mostly equivalent to the property graph model.
  - In a triple-store, all information is stored in the form of very simple three-part statements: (subject, predicate, object). The subject of a triple is equivalent to a vertex in a graph.
  - **The semantic web**: The triple-store data model is completely independent of the semantic web(e.g. Datomic)
  - **RDF(Resource Description Framework) data model**:
    - Tool – Apache Jena
- **Declarative query languages** for graphs: Cypher, SPARQL, and Datalog.
  - **The SPARQL("sparkle") query language**: SPARQL is a query language for triple-stores using the RDF data model
  - **Datalog**: much older language than SPARQL or Cypher, a foundation of later query language (e.g. Datomic, Cascalog),
    - Subset of Prolog.
    - Similar to the triple-store model, generalized a bit. Instead of writing a triple as (subject, predicate, object), we write it as predicate(subject, object).
- **Imperative graph query languages** such as Gremlin and graph processing frameworks like Pregel

Summary

- New non-relational "NoSQL" datastores have **diverged** in **two main directions**:
  - **Document databases** target use cases:
    - where data comes in self-contained documents;
    - relationships between one document and another are rare.
  - **Graph databases** go in the opposite direction, targeting use cases:
    - where anything is potentially related to everything.
- **Document** and **Graph databases** typically **don't enforce a schema** for the data they store, which can make it easier to adapt applications to changing requirements
- schema is **explicit** (enforced on write) or **implicit** (handled on read).

《Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems》 https://amzn.to/2WYphy6

PREVIOUS POST
Designing Data-Intensive Applications – Chapter 1: Reliable, Scalable, and Maintainable Applications

NEXT POST
Designing Data-Intensive Applications – Chapter 4: Encoding and Evolution

Search …

CONTACT FORM

RECENT POSTS

2021.2.15 – CH – 习惯的分享(室)

CATEGORIES

[Eat-吃] (3)

[Exercise-动] (6)